An Approach to Early Fault Prediction in Software Systems Using K-Means Clustering

Raman Goyal

Deptt. Of CSE LLRIET Moga, India

Abstract— Quality of a software component can be measured in terms of fault proneness of data. Quality estimations are made using fault proneness data available from previously developed similar type of projects and the training data consisting of software measurements. To predict faulty modules in software data different techniques have been proposed which includes statistical method, machine learning methods, neural network techniques and clustering techniques. Predicting faults early in the software life cycle can be used to improve software process control and achieve high software reliability. The aim of proposed approach is to investigate that whether metrics available in the early lifecycle (i.e. requirement metrics), metrics available in the late lifecycle (i.e. code metrics) and metrics available in the early lifecycle (i.e. requirement metrics) combined with metrics available in the late lifecycle (i.e. code metrics) can be used to identify fault prone modules using decision tree based Model in combination of K-means clustering as preprocessing technique. This approach has been tested with CM1 real time defect datasets of NASA software projects. The high accuracy of testing results show that the proposed Model can be used for the prediction of the fault proneness of software modules early in the software life cycle.

Keywords- Clustering, Decision Tree, K-means, software quality.

I. INTRODUCTION

A software fault is a defect that causes software failure in an executable product. In software engineering, the non-conformance of software to its requirements is commonly called a bug. Software Engineers distinguish between software faults, software failures and software bugs. In case of a failure, the software does not do what the user expects but on the other hand fault is a hidden programming error that may or may not actually manifest as a failure and the non-conformance of software to its requirements is commonly called a bug.

To predict faults different metric measures are available. Metrics available during static code development such as Halstead complexity, McCabe complexity, can be used to check modules for fault proneness. Fenton offers an example where the same program functionality is achieved using different programming language constructs resulting in different static measures for that module. Fenton uses this example to argue the uselessness of static code attributes [4]. Therefore, using static features alone can be uninformative.

A module currently under development is fault prone if a module with the similar product or process metrics in an earlier project developed in the same environment was fault prone. Therefore, the information available early within the current project or from the previous project can be used in making predictions. Although requirements metrics alone are not best at finding the faults but these metrics can increase the performance prediction.

It is required that fault-prone prediction models should be efficient and accurate. Thus, combinations of static features extracted from requirements and code can be exceptionally good predictors for identifying modules that actually contains faults. [2]

To predict the fault in software data a variety of techniques have been proposed which includes statistical method, machine learning methods, neural network techniques and clustering techniques. Much of the work has done on checking the quality with statistical methods and machine learning methods using either requirement or static code metrics. But very few have worked on finding the faulty and non faulty modules by using clustering techniques. However, estimating the quality using clustering techniques on fusion of requirement and code metric has not been proposed in the literature. Prediction of fault-prone modules provides one way to support software quality engineering through improved scheduling and project control.

Clustering is an approach that uses software measurement data consisting of limited or no fault-proneness data for analyzing software quality. In this study, K-Means clustering algorithm is being used for the preprocessing of the defect data for decision tree based predictive models to predict faulty/non faulty modules. K-means classify data in to different k groups, k being a positive integer, based on the attributes or some features.

Grouping of data is done on the basis of minimizing sum of squares of distances between data and their cluster centroid. This approach is an iterative scheme which allows classification of data in to faulty and non-faulty modules based on Euclidean distance between data points.

Finally, ROC Curve has been plotted using the recall value that is used for evaluate the results.

I. METHODOLOGY

Methods for identifying fault-prone software modules support helps to improve resource planning and scheduling as well as facilitating cost avoidance by effective verification. Such models can be used to predict the response variable which can either be the class of a module (e.g. fault-prone or not fault-prone) or a quality factor (e.g. number of faults) for a module. However various software engineering practices limit the availability of fault proneness data. With its effect supervised clustering is not possible to implement for analyzing the quality of software. This limited fault proneness data can be clustered by using semi supervised clustering. It is a constraint based clustering scheme that uses software engineering expert's domain knowledge to iteratively create clusters as either fault-free or fault-prone [3]. Software quality prediction models seek to predict quality factors such as whether a component is fault prone or not

First of all, the data metrics of requirement phase and structural code attributes of software systems has been collected from NASA Metrics Data Program (MDP) data repository [1]. MDP repository contains data of 13 projects out of which only 3 projects data include requirement metrics. These three projects are CM1 (written in C), containing approximately 505 modules, JM1 (written in C++), containing 10878 modules, PC1 (written in C), containing 1107 modules. CM1 is a NASA spacecraft instrument, JM1 is a real time ground system and PC1 is an earth orbiting satellite system. After collecting and refining the CM1 data from NASA projects, requirement metrics and module metrics of the above projects have been joined by using inner join method [2] as shown in figure 1. Combining requirements and code metrics have done using inner join database operation. An inner join creates a new result table by combining column values of two tables (A and B) based upon the join-predicate. The query compares each row of A with each row of B to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row. The result of the join can be defined as the outcome of first taking the Cartesian product (or cross-join) of all records in the tables (combining every record in table A with every record in table B) - then return all records which satisfy the join predicate. Application of this operation to join requirement and code is possible using three metrics product_module_metrics, product_requirement_metrics and product_requirement_realtion. Join operation is performed on product_module_metrics and product_requirement_relation using common attribute Module ID and result is stored in temporary table. Then taking all records from temporary table and joining with product_requirement_metrics using Requirement ID [2]. In this paper only the CM1 dataset is used for the empirical study.

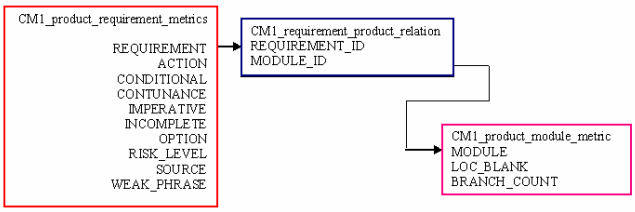

Figure1. E-R Diagram relates project requirements to modules and modules to faults

Clustering is a technique that divides data into two or more clusters depending upon some criteria. As, in this study data is divided into two clusters depending upon that whether they are fault free or fault prone. Then, K-means Clustering Algorithm is applied on the dataset and that clusters the software components into different clusters. Means the cluster Model is generated as Output of that step.

Thereafter, an operator is applied that clusters an exampleset given a cluster model. If an exampleSet does not contain id attributes it is probably not the same as the cluster model has been created on. Since cluster models depend on a static nature of the id attributes, the outcome on another exampleset with different values but same ids will be unpredictable. Only Centroid based clustering support assiging unseen examples to clusters.

Once we got the expanded dataset with additional meta data generated by the operator we applied decision Tree C4.5 algorithm and 10-fold cross validation is used to measure the performance of the built up model.

To predict the results, we have used confusion matrix. The confusion matrix has four categories: True positives (TP) are the modules correctly classified as faulty modules. False positives (FP) refer to fault-free modules incorrectly labeled as faulty. True negatives (TN) are the fault-free modules correctly labeled as such. False negatives (FN) refer to faulty modules incorrectly classified as fault-free modules.

TABLE I. A CONFUSION MATRIX OF PREDICTION OUTCOMES

Predicted	**Real data**		
		Fault	No Fault
	Fault	TP	FP
	No Fault	FN	TN

The following set of evaluation measures are being used to find the results:

- Probability of Detection (PD), also called recall or specificity, is defined as the probability of correct classification of a module that contains a fault.

$$PD = TP / (TP + FN) \quad (1)$$

- Probability of False Alarms (PF) is defined as the ratio of false positives to all non defect modules.

$$PF = FP / (FP + TN) \quad (2)$$

Basically, PD should be maximum and PF should be minimum [2].

A. ROC Curve

ROC is an acronym for Receiver Operator Characteristic; it is a plot of PD versus PF across all possible experimental combinations. ROC analysis is related in a direct and natural way to cost/benefit analysis of software projects by comparing their detected defective and non defective modules. ROC curve provides better insight to comparisons between different parametric values. The values of PD and PF ranges from 0 to 1. The graph is plotted along X-axis and Y-axis ranging from 0 to 1 both. The points plotted in ROC curve represent the visual comparison of classification performance of predictive models.

Typical ROC curve has concave shape with (0, 0) as the beginning and (1, 1) as the end. ROC curve is divided in to four different regions namely, cost adverse region, risk adverse region, negative curve and no information region.

Negative curve in ROC plot comprises data points with low PD and high PF. In software engineering practices, High PD and low PF specify that maximum number of modules has been classified correctly. As the PD decreases and PF increases, the probability that modules can be classified incorrectly increases. But this is not necessarily a bad news because it can be transposed into a preferred curve when classifier negates their tests.

II. IMPLEMENTATION & RESULTS

The NASA MDP datasets are used in this approach to estimate the quality of a software product. The datasets used is CM1 as this is one of the three projects in MDP data repository that include requirement metrics. We used requirement metrics, code metrics and join the requirement and code metrics as specified in [2] for modeling. There are 31 input attributes as metrics in the joined dataset created as mentioned in the Table 1. The 32nd column of the dataset is regarding the presence or absence of the fault in the software module. The resulting dataset after its polishing consists of 266 examples.

When the dataset is imported in the Rapidminer the snapshot of the dataview and meta data view are shown in figure 2 and figure 3 respectively.

A Cluster Model is created by applying K-means Clustering algorithm with the following parameters:

- Number of Clusters : 2
- Maximum Runs : 10
- Max. Optimization steps : 100

The dataset is partoned into two clusters: Cluster 0 and Cluster 1. The Cluster 0 consists of 259 items and Cluster 1consists of 7 items. The values of the two centroids are shown in Table III.

Thereafter, dataset is again reconstructed with the additional meta data information and dataset is given to the Decision Tree Algorithm C 4.5 with the following parameters:

- minimal_size_for_split : 4
- minimal_leaf_size : 2
- minimal_gain : 0.1
- maximal_depth : 20
- confidence : 0.25

TABLE II. LIST OF FUSION OF REQUIREMENT AND MODULE METRICS

Sr. No.	List of Metrics
1	ACTION
2	CONDITIONAL
3	CONTINUANCE
4	IMPERATIVE
5	INCOMPLETE
6	OPTION
7	RISK_LEVEL
8	WEAK_PHRASE
9	SOURCE
10	LOC_BLANK
11	BRANCH_COUNT
12	LOC_CODE_AND_COMMENT
13	LOC_COMMENTS
14	CYCLOMATIC_COMPLEXITY
15	DESIGN_COMPLEXITY
16	ESSENTIAL_COMPLEXITY
17	LOC_EXECUTABLE
18	HALSTEAD_CONTENT
19	HALSTEAD_DIFFICULTY
20	HALSTEAD_EFFORT
21	HALSTEAD_ERROR_EST
22	HALSTEAD_LENGTH
23	HALSTEAD_LEVEL
24	HALSTEAD_PROG_TIME
25	HALSTEAD_VOLUME
26	NUM_OPERANDS

27	NUM_OPERATORS
28	NUM_UNIQUE_OPERANDS
29	NUM_UNIQUE_OPERATORS
30	NUMBER_OF_LINES
31	LOC_TOTAL

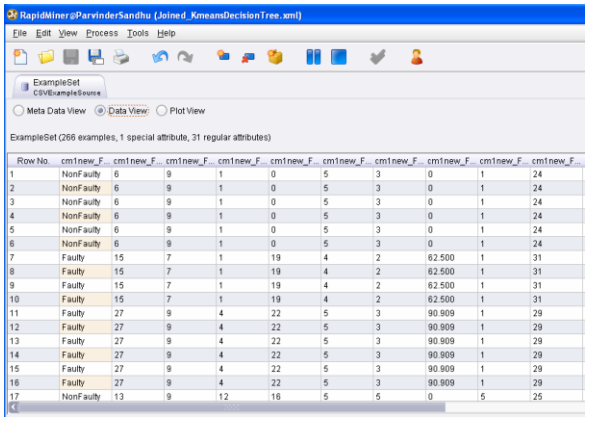

Figure2. Snapshot of the Data View of Dataset Imported in Rapidminer

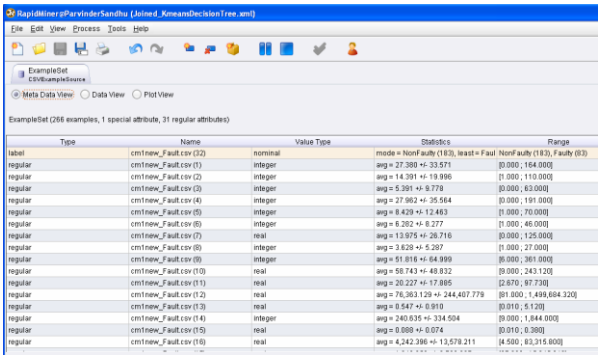

Figure3. Snapshot of the Meta Data View of Dataset Imported in Rapidminer

As shown in figure 4, the decision tree created is only based on 7th input attribute i.e. Risk_Level metric.

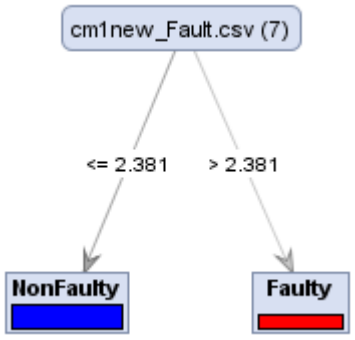

Figure 4. Snapshot of the Decision Tree

The generated decision tree model is evaluated with the 10-fold cross validation technique and the results are shown in table IV.

TABLE III. VALUE OF THE CENTROIDS AFTER CLUSTERING

Attribute	cluster_0	cluster_1
cm1new_Fault.csv (1)	23.687	164
cm1new_Fault.csv (2)	11.807	110
cm1new_Fault.csv (3)	4.537	37
cm1new_Fault.csv (4)	23.556	191
cm1new_Fault.csv (5)	6.764	70
cm1new_Fault.csv (6)	5.208	46
cm1new_Fault.csv (7)	14.353	0
cm1new_Fault.csv (8)	2.996	27
cm1new_Fault.csv (9)	43.459	361
cm1new_Fault.csv (10)	56.087	157.030
cm1new_Fault.csv (11)	18.132	97.730
cm1new_Fault.csv (12)	37,894.988	1,499,684.3
cm1new_Fault.csv (13)	0.424	5.120
cm1new_Fault.csv (14)	197.301	1844
cm1new_Fault.csv (15)	0.091	0.010
cm1new_Fault.csv (16)	2,105.277	83,315.800
cm1new_Fault.csv (17)	1,271.896	15,345.640
cm1new_Fault.csv (18)	73.772	711
cm1new_Fault.csv (19)	123.529	1133
cm1new_Fault.csv (20)	38.649	251
cm1new_Fault.csv (21)	19.734	69
cm1new_Fault.csv (22)	47.996	398
cm1new_Fault.csv (23)	1.471	1.286
cm1new_Fault.csv (24)	0.097	0
cm1new_Fault.csv (25)	0.386	0.429
cm1new_Fault.csv (26)	1.158	1
cm1new_Fault.csv (27)	0	0
cm1new_Fault.csv (28)	0.015	0
cm1new_Fault.csv (29)	1.656	1.857
cm1new_Fault.csv (30)	1.544	2
cm1new_Fault.csv (31)	0.069	0.143

TABLE IV. 10 FOLD CROSS VAILDATION RESULTS

	TRUE NONFAULTY	TRUE FAULTY	CLASS PRECISION
PREDICTED NONFAULTY	18	0	100.00%
PREDICTED FAULTY	0	9	100.00%
CLASS RECALL	100.00%	100.00%	

The value of TN is 18, TP is 9, FP is 0 and FN is 0. Hence, the PD and PF values are 1 and 0 respectively.

The values of the PD and PF i.e. (0,1) is plotted on the Receiver Operator Characteristic (ROC) as shown in the figure 5.

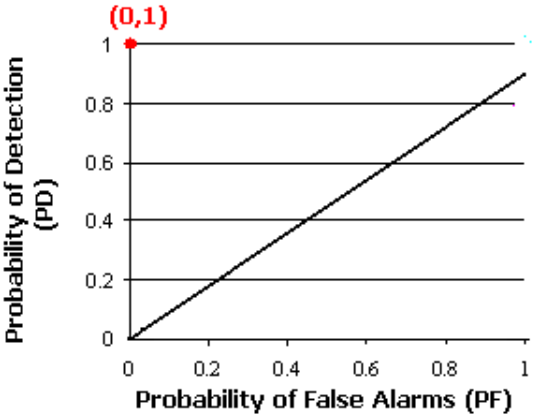

Figure 5. ROC Curve

III. CONCLUSION

This approach has been tested In this study it is tried to predict fault proneness of defect data at early stages of lifecycle combined with data available during code can help the project managers to build the projects with more accuracy and it will reduce the testing efforts as faulty areas are already predicted, so these modules can be handled properly. Here, a hybrid model consisting of K-means Clustering Algorithm and C 4.5 based Decision Tree is experimented with modified CM1 real time defect datasets of NASA software projects. The cross validation results shows 100% Class Precision and 100% Class Recall. When the results are plotted in the ROC curve then the optimal results are found in the preferred region. Hence, the high accuracy of testing results shows that the proposed Model can be used for the prediction of the fault proneness of software modules early in the software life cycle.

REFERENCES

[1] NASA IV &V Facility. Metric Data Program. Available from http: //MDP.ivv.nasa.gov/.

[2] Jiang Y., Cukic B. and Menzies T. (2007), "Fault Prediction Using Early Lifecycle Data". ISSRE 2007, the 18th IEEE Symposium on Software Reliability Engineering, IEEE Computer Society, Sweden, pp. 237-246.

[3] Seliya N., Khoshgoftaar T.M. and Zhong S. (2005), "Analyzing software quality with limited fault-proneness defect data", in proceedings of the Ninth IEEE international Symposium on High Asssurance System Engineering, Germany, pp. 89-98.

[4] Fenton N.E. and Pfleeger S.L. (1997), "Software Metrics: A Rigorous and Practical Approach". PWS publishing Company: ITP, Boston, MA, 2nd edition, pp.132-145.

[5] Munson J.C. and Khoshgoftaar T.M. (1992), "The detection of fault-prone programs", IEEE Transactions on Software Engineering, vol. 18, issue: 5, pp. 423-433.

[6] Bellini P. (2005), "Comparing Fault-Proneness Estimation Models", 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05), China, pp. 205-214.

[7] Lanubile F., Lonigro A., and Visaggio G. (1995) "Comparing Models for Identifying Fault-Prone Software Components", Proceedings of Seventh International Conference on Software Engineering and Knowledge Engineering, USA, pp. 12-19.

[8] Fenton N.E. and Neil M. (1999), "A Critique of Software Defect Prediction Models", IEEE Transactions on Software Engineering, vol. 25, issue: 5, pp. 675-689.

[9] Runeson, Wohlin C. and Ohlsson M.C. (2001), "A Proposal for Comparison of Models for Identification of Fault-Proneness", Journal of System and Software, vol. 56, issue: 3, pp. 301–320

[10] Runeson, Wohlin C. and Ohlsson M.C. (2001), "A Proposal for Comparison of Models for Identification of Fault-Proneness", Journal of System and Software, vol. 56, issue: 3, pp. 301–320.

[11] Challagulla V.U.B., Bastani F.B., Yen I. L. and Paul (2005) "Empirical assessment of machine learning based software defect prediction techniques", 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, USA, pp. 263-270.

[12] Basu S., Banerjee A. and Moorey R.. (2002) "Semi-Supervised Clusering by Seeding". In Proceedings of the 19th International Conference on Machine Learning, Sydney, pp. 19-26

[13] Brodely C.E. and Friedl. M.A. (1999) "Identifying mislabeled training Data". Journal of Artificial Intelligence Research, vol. 11, pp.131-167.

◈◈◈